Ensemble Learning

Nipun Batra and teaching staff

IIT Gandhinagar

August 20, 2025

Table of Contents

1. Introduction to Ensemble Learning
2. Bagging: Bootstrap Aggregation
3. Boosting: Learning from Mistakes
4. Random Forest: Double Randomness

Introduction to Ensemble Learning

What is Ensemble Learning?

Important: The Core Idea

"The wisdom of crowds": Combine multiple models to make better predictions than any single model could achieve alone.

Key Points

Key Insight:

- Individual models make different mistakes
- By combining them intelligently, we can reduce overall error
- Most Kaggle competition winners use ensemble methods!

Real-World Analogy: Medical Diagnosis

Example: Why Do We Seek Second Opinions?

Single Doctor:

- Might miss subtle symptoms
- Could have personal biases
- Limited by individual experience

Multiple Doctors (Ensemble):

- Different perspectives and expertise
- Collective wisdom reduces misdiagnosis
- More robust and reliable decisions

Simple Ensemble Examples

Example: Classification

Problem: Spam detection
Individual Predictions:

- Model 1: Spam
- Model 2: Spam
- Model 3: Not Spam

Ensemble (Majority Vote): Spam

Example: Regression

Problem: House price prediction
Individual Predictions:

- Model 1: $420K
- Model 2: $450K
- Model 3: $430K

Ensemble (Average): $433K

Why Do Ensembles Work? Three Key Reasons

Important: Based on Ensemble Methods in ML by Dietterich

Three fundamental reasons why combining models works better:

Key Points

1. **Statistical**: Limited data → Multiple valid hypotheses
2. **Computational**: Models get stuck in local optima
3. **Representational**: Individual models have limitations

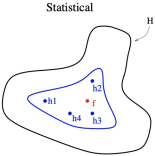
Reason 1: Statistical Problem

Definition: The Statistical Challenge

When data is limited, many competing hypotheses can achieve the same accuracy on training data.

Example: Decision Trees Example

- Same dataset → multiple valid trees
- Combining reduces risk of picking the “wrong” one



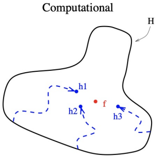
Reason 2: Computational Problem

Definition: The Computational Challenge

Learning algorithms can get stuck in local optima or use greedy strategies.

Example: Examples

- Decision trees: greedy splits
- Neural networks: local minima
- Different runs → different solutions



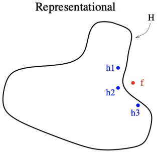
Reason 3: Representational Problem

Definition: The Representational Challenge

Some models cannot learn the true form of the target function.

Example: Limitations

- Decision trees: axis-parallel splits only
- Linear models: no non-linear relationships
- Each model has inherent biases

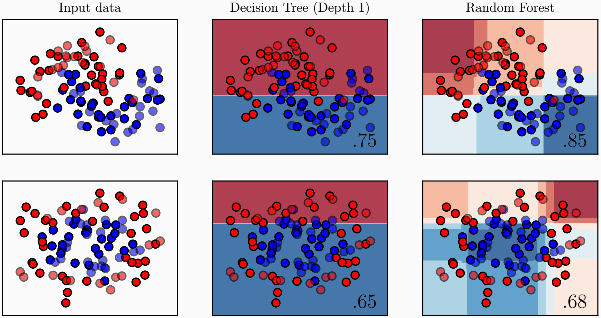


Visual Example: Decision Trees vs Random Forest

Important: Representation Comparison

Question: How do individual decision trees compare to their ensemble?

Notebook: ensemble-representation.html



Individual Tree Behavior

Key Points

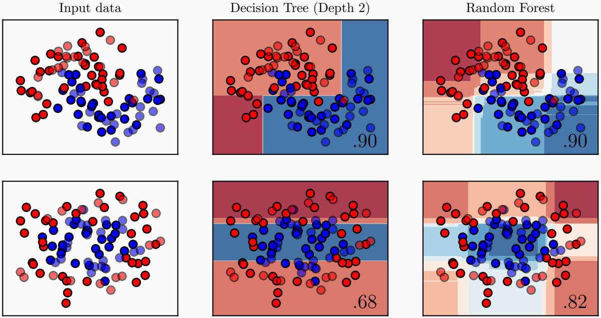
Observation: Individual trees create rigid, rectangular decision boundaries

Random Forest: The Power of Combination

Example: Ensemble Effect

Result: Combining multiple trees creates smoother, more flexible decision boundaries

Notebook: ensemble-representation.html



Ensemble Advantage

Key Points

Key Insight: The ensemble overcomes individual model limitations!

When Do Ensembles Work? Two Key Requirements

Definition: Necessary and Sufficient Conditions

For an ensemble to outperform individual members, models must be:

1. **Accurate**: Better than random guessing
2. **Diverse**: Make different errors on new data

Key Points

Key Terms:

- **Accurate**: Error rate $< 50\%$ (better than coin flip)
- **Diverse**: Models disagree on different examples

Diversity: The Magic Ingredient

Important: Identical Models (No Diversity)

Scenario: All three models make the same mistakes
When $h_1(x)$ is wrong:

- $h_2(x)$ is also wrong
- $h_3(x)$ is also wrong
- Ensemble prediction: Wrong

Example: Diverse Models

Scenario: Models make different mistakes
When $h_1(x)$ is wrong:

- $h_2(x)$ might be correct
- $h_3(x)$ might be correct
- Ensemble prediction: Correct!

Key Points

Bottom Line: Diversity allows the ensemble to correct individual model errors!

Mathematical Proof: Why Ensembles Work

Definition: Majority Voting Analysis

Setup: 3 models, each with error probability $\varepsilon = 0.3$
Ensemble fails when: 2 or 3 models are wrong

Example: Calculation

$$P(\text{ensemble wrong}) = \binom{3}{2}\varepsilon^2(1-\varepsilon) + \binom{3}{3}\varepsilon^3$$
$$= 3 \times 0.3^2 \times 0.7 + 1 \times 0.3^3$$
$$= 0.189 + 0.027 = \mathbf{0.216}$$

Key Points

Result: Ensemble error (21.6%) $<$ Individual error (30%)!

The Power of Scaling: More Models = Better Performance

Example: Good Individual Models ($\varepsilon = 0.3$)

# Models	Ensemble Error
1	30.0%
3	21.6%
5	16.3%

Ensembles help!

Important: Poor Individual Models ($\varepsilon = 0.6$)

# Models	Ensemble Error
1	60.0%
3	64.8%
5	68.3%

Ensembles hurt!

Key Points

Key Insight: Ensembles only help when base models are better than random!

When Ensembles Fail: Common Pitfalls

Important: Ensemble Limitations

Ensembles DON'T work well when:

Example: Poor Base Models

- Individual accuracy $< 50\%$
- Models worse than random guessing
- Garbage in $\rightarrow$ Garbage out

Example: Lack of Diversity

- All models make same mistakes
- High correlation between predictions
- No complementary strengths

Key Points

Solution: Ensure base models are accurate AND diverse!

Bagging: Bootstrap Aggregation

What is Bagging?

Definition: Bagging = Bootstrap + Aggregation

Goal: Create diverse models from a single dataset to reduce variance

Key Points

Key Insight: Even with the same algorithm and same data, we can create different models by training on different subsets!

Important: The Challenge

Problem: How do we get different training sets from one dataset?
Solution: Bootstrap sampling (sampling with replacement)

Bootstrap Sampling: The Core Technique

Example: Bootstrap Process

Original Dataset: $D = \{D_1, D_2, D_3, \ldots, D_n\}$
For each model: Create new dataset by sampling n examples *with replacement*

Definition: Bootstrap Sample 1

$D_1, D_3, D_6, D_1, D_5, \ldots$
Notice: D_1 appears twice!

Definition: Bootstrap Sample 2

$D_2, D_4, D_1, D_n, D_3, \ldots$
Different sample, different model!

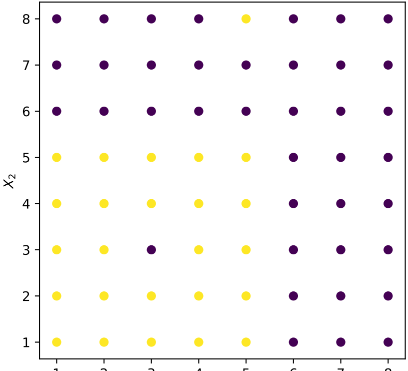
Key Points

Result: Each bootstrap sample is slightly different → Diverse models!

Bagging Example: The Dataset

Important: Classification Problem

Task: Classify points as red or blue circles
Challenge: Points (3,3) and (5,8) are outliers/anomalies



The Outlier Challenge

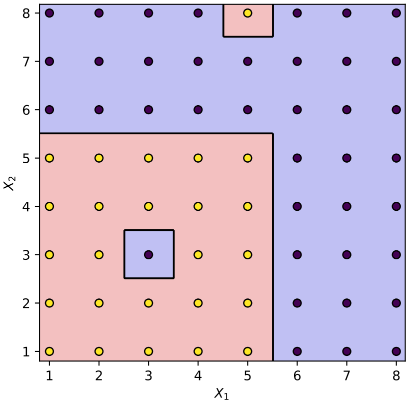
Key Points

Question: How will a single decision tree handle these outliers?

Single Decision Tree: Overfitting Problem

Example: Deep Decision Tree (Depth = 6)

Result: Complex boundary that memorizes outliers



The High Variance Problem

Important: The Problem

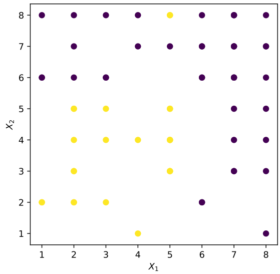
High Variance: Small changes in data → Very different decision boundaries

Bootstrap Samples: Part 1

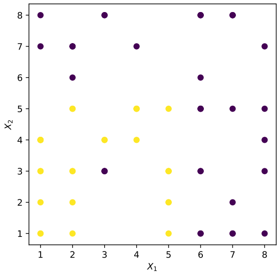
Example: Creating Diverse Training Sets

Generate different bootstrap samples from original dataset

Sample 1

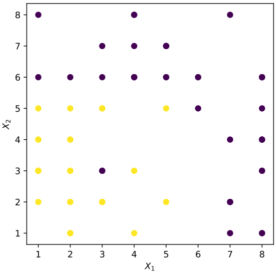


Sample 2

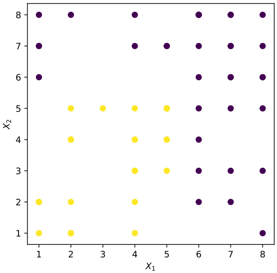


Bootstrap Samples: Part 2

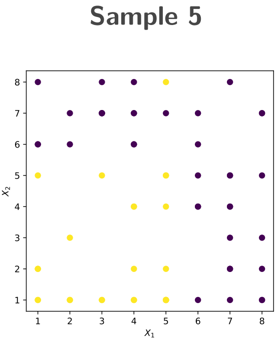
Sample 3



Sample 4



Bootstrap Samples: Part 3



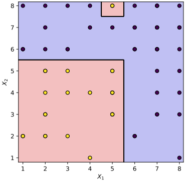
Key Points

Key Insight: Each sample has different combinations of points!

Bagging : Classification Example

Bagging : Classification Example

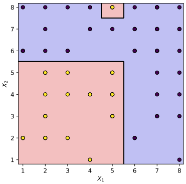
Round - 1



Tree Depth = 4

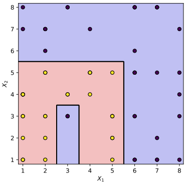
Bagging : Classification Example

Round - 1



Tree Depth $= 4$

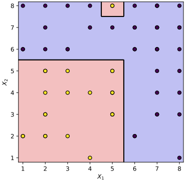
Round - 2



Tree Depth $= 5$

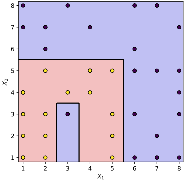
Bagging : Classification Example

Round - 1



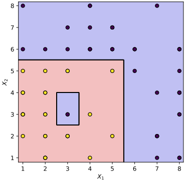
Tree Depth = 4

Round - 2



Tree Depth = 5

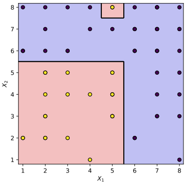
Round - 3



Tree Depth = 5

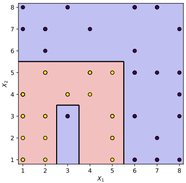
Bagging : Classification Example

Round - 1



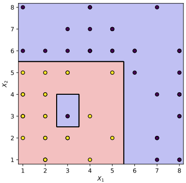
Tree Depth = 4

Round - 2



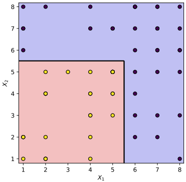
Tree Depth = 5

Round - 3



Tree Depth = 5

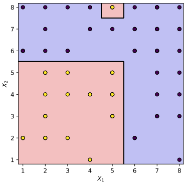
Round - 4



Tree Depth = 2

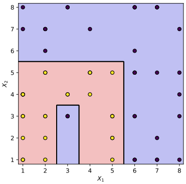
Bagging : Classification Example

Round - 1



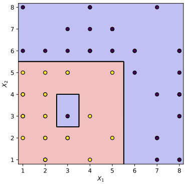
Tree Depth = 4

Round - 2



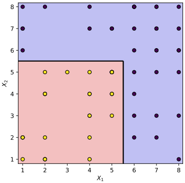
Tree Depth = 5

Round - 3



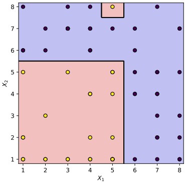
Tree Depth = 5

Round - 4



Tree Depth = 2

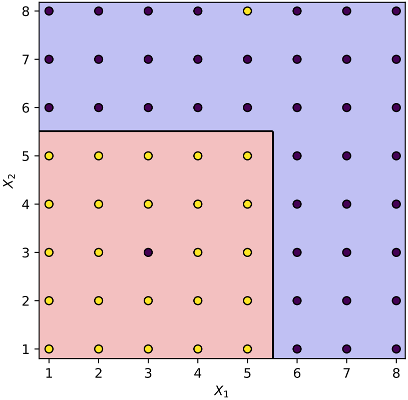
Round - 5



Tree Depth = 4

Bagging : Classification Example

Using majority voting to combine all predictions, we get the decision boundary below.



Bagging

Summary

- We take “strong” learners and combine them to reduce variance.
- All learners are independent of each other.

Boosting: Learning from Mistakes

What is Boosting?

Definition: Boosting Philosophy

Goal: Combine weak learners sequentially to create a strong ensemble

Key Points

Key Differences from Bagging:

- **Sequential:** Models built one after another (not in parallel)
- **Focus on Mistakes:** Each model learns from previous model's errors
- **Reduce Bias:** Turn weak learners into strong ensemble

Example: The Boosting Intuition

“If at first you don’t succeed, try harder on what you got wrong!”

Boosting vs Bagging: Side-by-Side Comparison

Important: Bagging

Strategy: Parallel learning

- Models trained independently
- Reduces variance
- Works with "strong" learners
- Bootstrap sampling
- Simple majority voting

Key Points

Boosting **Strategy:** Sequential learning

- Models learn from mistakes
- Reduces bias
- Works with "weak" learners
- Weighted sampling
- Weighted combination

Definition: Weak Learner Definition

Weak Learner: Any classifier that performs slightly better than random guessing (accuracy $>$ 50% for binary classification)

AdaBoost: Adaptive Boosting

Definition: AdaBoost Core Idea

Each model adapts to previous model's mistakes

Key Points

Process:

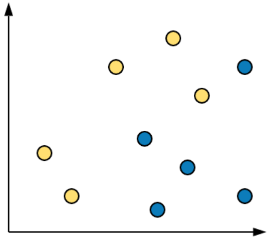
1. Train weak learner on weighted data
2. Increase weights of misclassified examples
3. Repeat: Focus on "hard" examples

AdaBoost Step-by-Step: Problem Setup

Definition: AdaBoost Notation

- N training samples: $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$
- Sample weights: w_i (importance of sample i for training)
- M weak learners: $h_1, h_2, \ldots, h_M$
- Learner weights: α_m (importance of learner m in final ensemble)

AdaBoost: Visual Setup



Key Points

Goal: Learn a strong classifier $H(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m h_m(x)\right)$

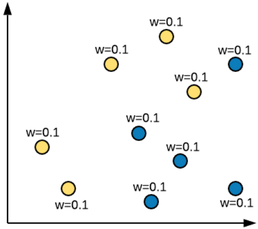
AdaBoost Step 1: Initialize Sample Weights

Important: Step 1: Equal Importance for All Samples

Initialize: $w_i^{(1)} = \frac{1}{N}$ for all $i = 1, 2, \ldots, N$

Why equal weights?

- No prior knowledge about “hard” samples
- Weights adapt as we learn from mistakes



AdaBoost Step 2: Train First Weak Learner

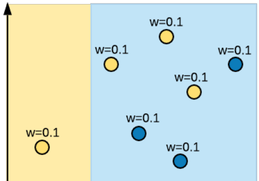
Important: Step 2a: Train Classifier on Weighted Data

Train weak learner h_1 using current sample weights $w_i^{(1)}$

Key Points

Key Insight: Higher weight = more training focus

- Initially all weights equal → standard training



AdaBoost Step 2: Evaluate First Classifier

Important: Step 2b: Identify Mistakes

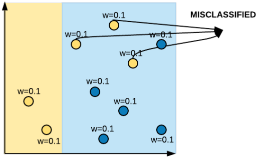
First classifier h_1 makes some mistakes (shown in red crosses)

Key Points

Important Observation:

- Even weak learners make mistakes
- These mistakes guide the next learning step
- Key question: How much do we trust this classifier?

AdaBoost Step 2: Visualizing Mistakes



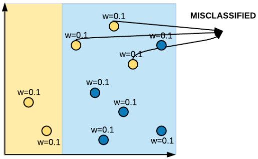
Example: Teacher Analogy

After first quiz, teacher sees which students got questions wrong

AdaBoost Step 3: Calculate Error

Definition: Weighted Error

$$\text{err}_m = \frac{\text{weights of mistakes}}{\text{total weights}}$$



Example: Example

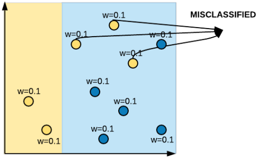
$err_1 = 0.3$ (30% error → better than random)

AdaBoost Step 4: Calculate Classifier Weight

Definition: Classifier Weight Formula

$$\alpha_m = \frac{1}{2} \ln \left(\frac{1 - \mathrm{err}_m}{\mathrm{err}_m} \right)$$

Purpose: Determine how much to trust this classifier in final ensemble



AdaBoost Step 4: Example Calculation

Example: Example Calculation

$err_1 = 0.3$
$\alpha_1 = \frac{1}{2} \ln\left(\frac{1-0.3}{0.3}\right) = \frac{1}{2}\ln(2.33) = 0.42$

Key Points

Alpha Intuition:

- Lower error → Higher α
- Higher α → More trust
- $\alpha = 0$ when $err = 0.5$ (random)

Understanding the Alpha Formula

Key Points

What does $\alpha = \frac{1}{2} \ln \left(\frac{1-err}{err} \right)$ **really mean?**

Definition: Perfect Classifier

$err = 0 \Rightarrow \alpha = +\infty$
Translation: Infinite trust

Example: Good Classifier

$err = 0.1 \Rightarrow \alpha = 1.1$
Translation: High trust

Important: Random Classifier

$err = 0.5 \Rightarrow \alpha = 0$
Translation: No trust

Important: Worse than Random

$err = 0.9 \Rightarrow \alpha = -1.1$
Translation: Negative trust (flip predictions!)

AdaBoost: Mathematical Elegance

Example: Key Insight

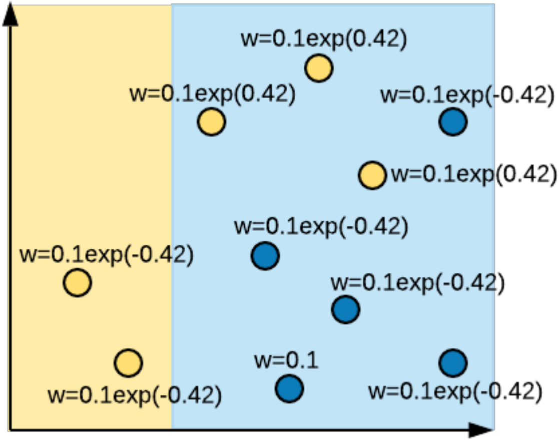
AdaBoost is mathematically elegant: Even “bad” classifiers (worse than random) can be useful by flipping their predictions!

Boosting : AdaBoost

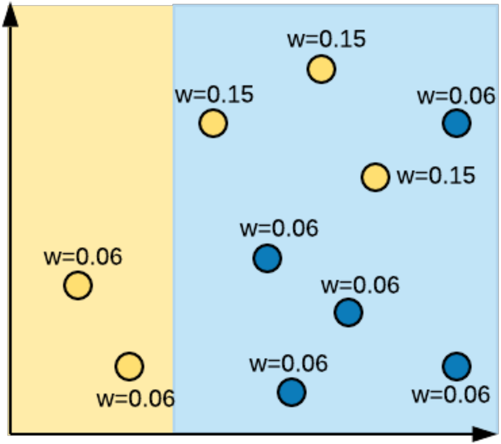
Consider we have a dataset of N samples.
Sample i has weight w_i. There are M classifiers in the ensemble.

1. Initialize weights of data samples: $w_i = \frac{1}{N}$
2. For $m = 1, \ldots, M$:
    1) Learn classifier using current weights w_i's
    2) Compute the weighted error: $\text{err}_m = \frac{\sum_i w_i(\text{incorrect})}{\sum_i w_i}$
    3) Compute $\alpha_m = \frac{1}{2} \log_e \left(\frac{1-\text{err}_m}{\text{err}_m} \right)$
    4) For samples which were predicted correctly: $w_i = w_i e^{-\alpha_m}$
    5) For samples which were predicted incorrectly: $w_i = w_i e^{\alpha_m}$

Boosting : AdaBoost



Boosting : AdaBoost

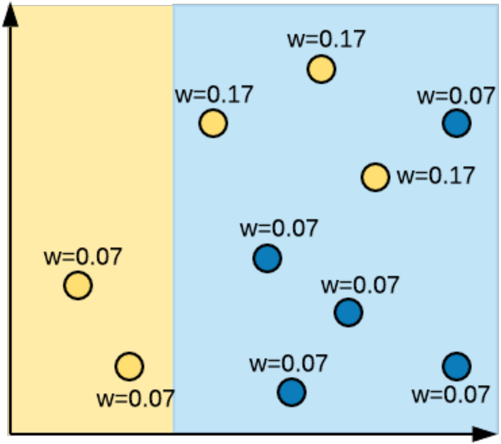


Boosting : AdaBoost

Consider we have a dataset of N samples.
Sample i has weight w_i. There are M classifiers in the ensemble.

1. Initialize weights of data samples: $w_i = \frac{1}{N}$
2. For $m = 1, \ldots, M$:
   1) Learn classifier using current weights w_i's
   2) Compute the weighted error: $\text{err}_m = \frac{\sum_i w_i(\text{incorrect})}{\sum_i w_i}$
   3) Compute $\alpha_m = \frac{1}{2} \log_e \left(\frac{1 - \text{err}_m}{\text{err}_m} \right)$
   4) For samples which were predicted correctly: $w_i = w_i e^{-\alpha_m}$
   5) For samples which were predicted incorrectly: $w_i = w_i e^{\alpha_m}$
   6) Normalize w_i's to sum to 1.

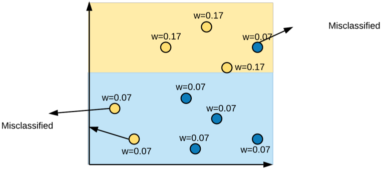
Boosting : AdaBoost



Boosting : AdaBoost

Consider we have a dataset of N samples.
Sample i has weight w_i. There are M classifiers in the ensemble.

1. Initialize weights of data samples: $w_i = \frac{1}{N}$
2. For $m = 1, \ldots, M$:
    1) Learn classifier using current weights w_i's
    2) Compute the weighted error: $\text{err}_m = \frac{\sum_i w_i(\text{incorrect})}{\sum_i w_i}$
    3) Compute $\alpha_m = \frac{1}{2}\log_e\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$



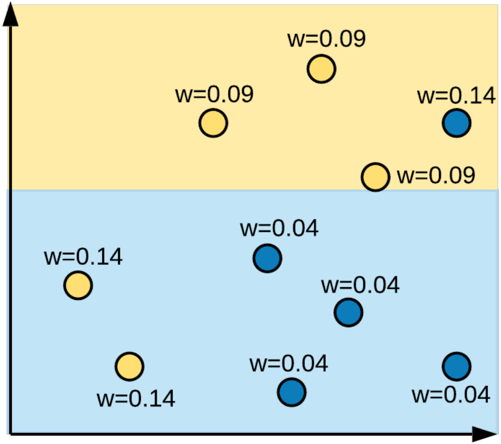
$$err_2 = \frac{0.21}{1}$$
$$\alpha_2 = \frac{1}{2}log\left(\frac{1-0.21}{0.21}\right) = 0.66$$

Boosting : AdaBoost

Consider we have a dataset of N samples.
Sample i has weight w_i. There are M classifiers in the ensemble.

1. Initialize weights of data samples: $w_i = \frac{1}{N}$
2. For $m = 1, \ldots, M$:
    1) Learn classifier using current weights w_i's
    2) Compute the weighted error: $\text{err}_m = \frac{\sum_i w_i(\text{incorrect})}{\sum_i w_i}$
    3) Compute $\alpha_m = \frac{1}{2} \log_e \left(\frac{1-\text{err}_m}{\text{err}_m} \right)$
    4) For samples which were predicted correctly: $w_i = w_i e^{-\alpha_m}$
    5) For samples which were predicted incorrectly: $w_i = w_i e^{\alpha_m}$

Boosting : AdaBoost

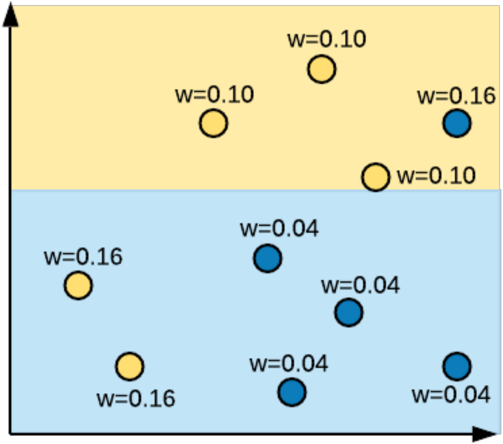


Boosting : AdaBoost

Consider we have a dataset of N samples.
Sample i has weight w_i. There are M classifiers in the ensemble.

1. Initialize weights of data samples: $w_i = \frac{1}{N}$
2. For $m = 1, \dots, M$:
   1) Learn classifier using current weights w_i's
   2) Compute the weighted error: $\mathrm{err}_m = \frac{\sum_i w_i(\text{incorrect})}{\sum_i w_i}$
   3) Compute $\alpha_m = \frac{1}{2} \log_e \left(\frac{1-\mathrm{err}_m}{\mathrm{err}_m} \right)$
   4) For samples which were predicted correctly: $w_i = w_i e^{-\alpha_m}$
   5) For samples which were predicted incorrectly: $w_i = w_i e^{\alpha_m}$
   6) Normalize w_i's to sum to 1.

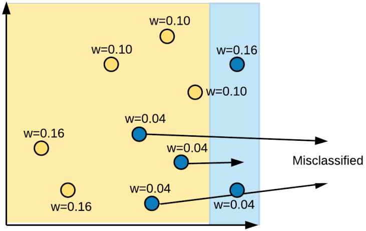
Boosting : AdaBoost



Boosting : AdaBoost

Consider we have a dataset of N samples.
Sample i has weight w_i. There are M classifiers in the ensemble.

1. Initialize weights of data samples: $w_i = \frac{1}{N}$
2. For $m = 1, \ldots, M$:
   1) Learn classifier using current weights w_i's
   2) Compute the weighted error: $\text{err}_m = \frac{\sum_i w_i(\text{incorrect})}{\sum_i w_i}$
   3) Compute $\alpha_m = \frac{1}{2}\log_e\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$



$$err_3 = \frac{0.12}{1}$$

$$\alpha_3 = \frac{1}{2} log\left(\frac{1-0.12}{0.12}\right) = 0.99$$

Boosting: Adaboost

Intuitively, after each iteration, importance of wrongly classified samples is increased by increasing their weights and importance of correctly classified samples is decreased by decreasing their weights.

Boosting: Adaboost

Testing

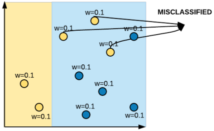
- For each sample x, compute the prediction of each classifier $h_m(x)$.
- Final prediction is the sign of the sum of weighted predictions, given as:
- $\mathrm{SIGN}(\alpha_1 h_1(x) + \alpha_2 h_2(x) + \ldots + \alpha_M h_M(x))$

Boosting: Adaboost

Example

Boosting: Adaboost

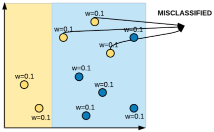
Example



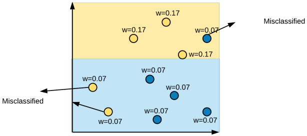
$\alpha_1 = 0.42$

Boosting: Adaboost

Example



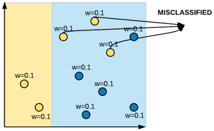
$\alpha_1 = 0.42$



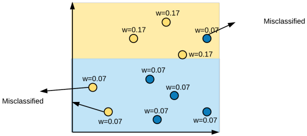
$\alpha_2 = 0.66$

Boosting: Adaboost

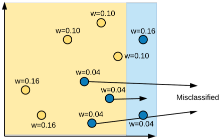
Example



$\alpha_1 = 0.42$



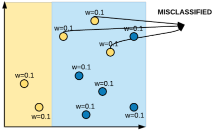
$\alpha_2 = 0.66$



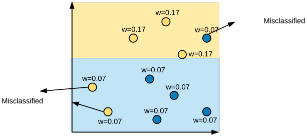
$\alpha_3 = 0.99$

Boosting: Adaboost

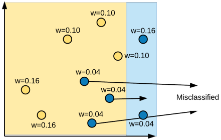
Example



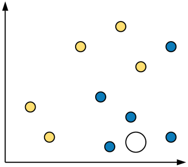
$\alpha_1 = 0.42$



$\alpha_2 = 0.66$

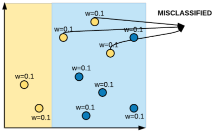


$\alpha_3 = 0.99$

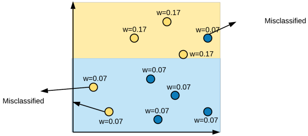


Boosting: Adaboost

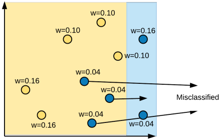
Example



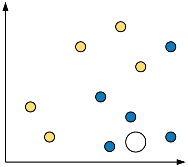
$\alpha_1 = 0.42$



$\alpha_2 = 0.66$



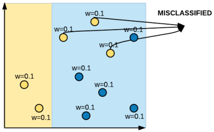
$\alpha_3 = 0.99$



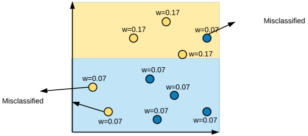
Let us say, yellow class is $+1$ and blue class is -1

Boosting: Adaboost

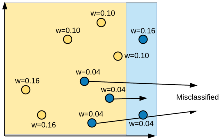
Example



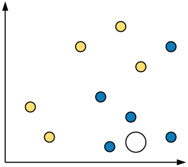
$\alpha_1 = 0.42$



$\alpha_2 = 0.66$



$\alpha_3 = 0.99$

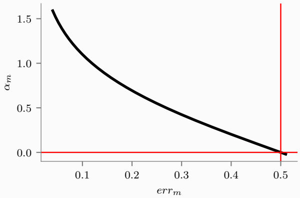


Let us say, yellow class is +1 and blue class is -1
Prediction = SIGN(0.42*-1 + 0.66*-1 + 0.99*+1) = Negative = blue

Intuition behind weight update formula

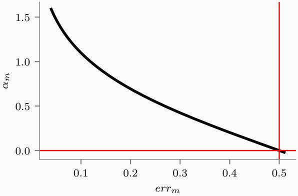
Intuition behind weight update formula

Notebook: boosting-explanation.html

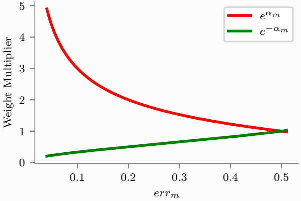


Intuition behind weight update formula

Notebook: boosting-explanation.html

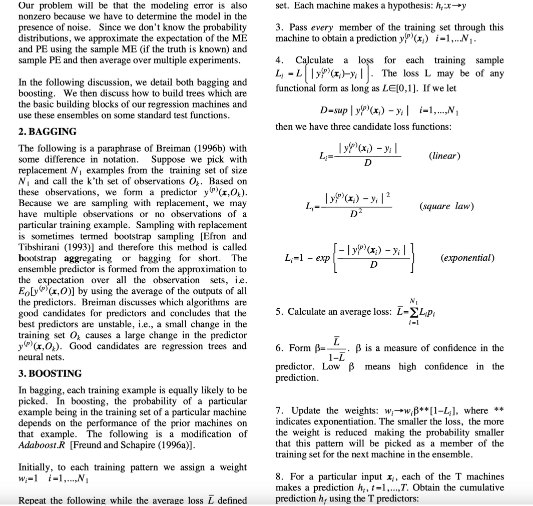


Notebook: boosting-explanation.html



ADABoost for regresion

From Paper: Improving Regressors using Boosting Techniques



Our problem will be that the modeling error is also nonzero because we have to determine the model in the presence of noise. Since we don't know the probability distributions, we approximate the expectation of the ME and PE using the sample ME (if the truth is known) and sample PE and then average over multiple experiments.

In the following discussion, we detail both bagging and boosting. We then discuss how to build trees which are the basic building blocks of our regression machines and use these ensembles on some standard test functions.

2. BAGGING

The following is a paraphrase of Breiman (1996b) with some difference in notation. Suppose we pick with replacement N_1 examples from the training set of size N_1 and call the k'th set of observations O_k. Based on these observations, we form a predictor $y^{(p)}(\boldsymbol{x},O_k)$. Because we are sampling with replacement, we may have multiple observations or no observations of a particular training example. Sampling with replacement is sometimes termed bootstrap sampling [Efron and Tibshirani (1993)] and therefore this method is called **b**ootstrap **agg**regating or bagging for short. The ensemble predictor is formed from the approximation to the expectation over all the observation sets, i.e. $E_O[y^{(p)}(\boldsymbol{x},O)]$ by using the average of the outputs of all the predictors. Breiman discusses which algorithms are good candidates for predictors and concludes that the best predictors are unstable, i.e., a small change in the training set O_k causes a large change in the predictor $y^{(p)}(\boldsymbol{x},O_k)$. Good candidates are regression trees and neural nets.

3. BOOSTING

In bagging, each training example is equally likely to be picked. In boosting, the probability of a particular example being in the training set of a particular machine depends on the performance of the prior machines on that example. The following is a modification of *Adaboost.R* [Freund and Schapire (1996a)].

Initially, to each training pattern we assign a weight $w_i=1 \quad i=1,\ldots,N_1$

Repeat the following while the average loss $\bar{L}$ defined

set. Each machine makes a hypothesis: $h_t:x \rightarrow y$

3. Pass *every* member of the training set through this machine to obtain a prediction $y_i^{(p)}(\boldsymbol{x}_i) \quad i=1,\ldots N_1$.

4. Calculate a loss for each training sample $L_i = L\left[\, |y_i^{(p)}(\boldsymbol{x}_i)-y_i| \,\right]$. The loss L may be of any functional form as long as $L\in[0,1]$. If we let

$$D=sup\,|y_i^{(p)}(\boldsymbol{x}_i)-y_i| \quad i=1,\ldots,N_1$$

then we have three candidate loss functions:

$$L_i=\frac{|y_i^{(p)}(\boldsymbol{x}_i)-y_i|}{D} \qquad (linear)$$

$$L_i=\frac{|y_i^{(p)}(\boldsymbol{x}_i)-y_i|^2}{D^2} \qquad (square\ \ law)$$

$$L_i=1-exp\left[\frac{-|y_i^{(p)}(\boldsymbol{x}_i)-y_i|}{D}\right] \qquad (exponential)$$

5. Calculate an average loss: $\bar{L}=\sum_{i=1}^{N_1} L_i p_i$

6. Form $\beta=\frac{\bar{L}}{1-\bar{L}}$. β is a measure of confidence in the predictor. Low β means high confidence in the prediction.

7. Update the weights: $w_i \rightarrow w_i\beta{**}[1-L_i]$, where ** indicates exponentiation. The smaller the loss, the more the weight is reduced making the probability smaller that this pattern will be picked as a member of the training set for the next machine in the ensemble.

8. For a particular input $\boldsymbol{x}_i$, each of the T machines makes a prediction h_t, $t=1,\ldots,T$. Obtain the cumulative prediction h_f using the T predictors:

Random Forest: Double Randomness

What is Random Forest?

Definition: Random Forest = Bagging + Feature Randomness

Core Idea: Combine many decision trees trained on random subsets of data AND random subsets of features

Key Points

Two Sources of Randomness:

- **Bootstrap Sampling:** Each tree sees different training samples (like bagging)
- **Feature Subsampling:** Each split considers only random subset of features

Why Double Randomness?

Example: Why Double Randomness?

Goal: Create diverse trees that make different mistakes

- Data randomness → Different perspectives on the problem
- Feature randomness → Different decision criteria
- Result: Highly decorrelated predictions!

Random Forest Algorithm: Hyperparameters

Definition: Random Forest Hyperparameters

- B = Number of trees in the forest
- m = Number of features considered at each split (typically $\sqrt{M}$ or $\log_2(M)$)
- max_depth = Maximum depth of each tree

Random Forest Algorithm: Training Process

Important: Random Forest Training Algorithm

For $b = 1, 2, \ldots, B$:

1. **Bootstrap:** Sample n training examples with replacement
2. **Train Tree:** Build tree with feature randomness:
   - At each split: randomly select m out of M features
   - Choose best split among these m features only

Key Points

Key Insight: Each tree is "strong" individually but they make different mistakes due to randomness!

Random Forest: Feature Selection at Each Split

Example: Iris Example

4 features available $\Rightarrow m = \sqrt{4} = 2$ features per split

Definition: Tree 1 - Split 1

Random subset: {Sepal Length, Petal Width}
Best split: Petal Width $<$ 0.8

Definition: Tree 2 - Split 1

Random subset: {Sepal Width, Petal Length}
Best split: Petal Length $<$ 2.5

Random Forest: The Power of Feature Diversity

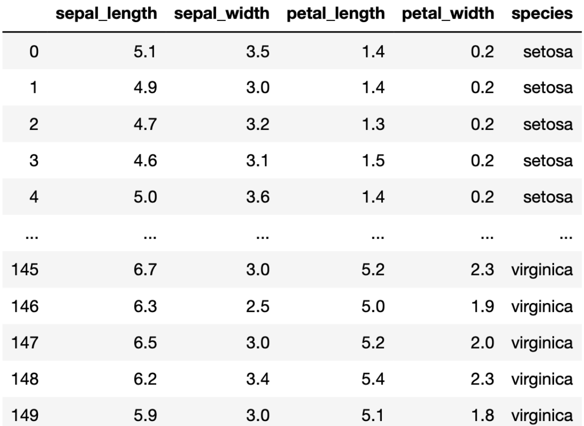
Key Points

Result: Different trees focus on different feature combinations → Diverse predictions

Random Forest Example: Iris Dataset

Example: The Iris Classification Problem

Task: Classify iris flowers into 3 species based on 4 measurements



	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

Random Forest Setup for Iris

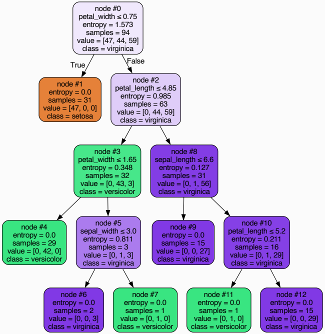
Key Points

Random Forest Setup:

- 4 features available → Consider $m = 2$ features per split
- Bootstrap samples for each tree
- Combine predictions via majority voting

Decision Tree # 0

Notebook: ensemble-feature-importance.html



node #0
petal_width ≤ 0.75
entropy = 1.573
samples = 94
value = [47, 44, 59]
class = virginica

True

False

node #1
entropy = 0.0
samples = 31
value = [47, 0, 0]
class = setosa

node #2
petal_length ≤ 4.85
entropy = 0.985
samples = 63
value = [0, 44, 59]
class = virginica

node #3
petal_width ≤ 1.65
entropy = 0.348
samples = 32
value = [0, 43, 3]
class = versicolor

node #8
sepal_length ≤ 6.6
entropy = 0.127
samples = 31
value = [0, 1, 56]
class = virginica

node #4
entropy = 0.0
samples = 29
value = [0, 42, 0]
class = versicolor

node #5
sepal_width ≤ 3.0
entropy = 0.811
samples = 3
value = [0, 1, 3]
class = virginica

node #9
entropy = 0.0
samples = 15
value = [0, 0, 27]
class = virginica

node #10
petal_length ≤ 5.2
entropy = 0.211
samples = 16
value = [0, 1, 29]
class = virginica

node #6
entropy = 0.0
samples = 2
value = [0, 0, 3]
class = virginica

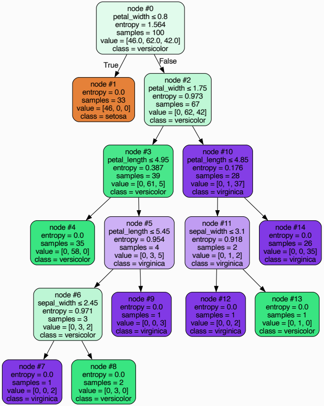
node #7
entropy = 0.0
samples = 1
value = [0, 1, 0]
class = versicolor

node #11
entropy = 0.0
samples = 1
value = [0, 1, 0]
class = versicolor

node #12
entropy = 0.0
samples = 15
value = [0, 0, 29]
class = virginica

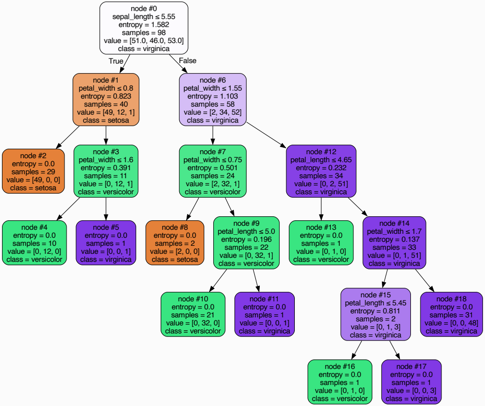
Decision Tree # 1

Notebook: ensemble-feature-importance.html



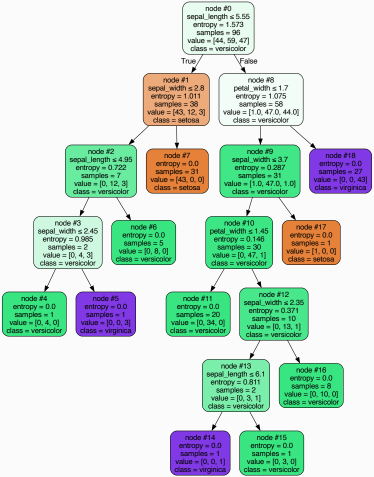
Decision Tree # 2

Notebook: ensemble-feature-importance.html



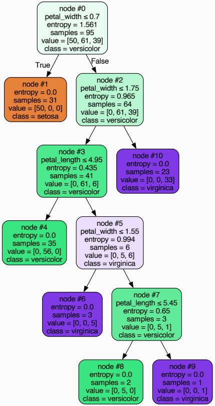
Decision Tree # 3

Notebook: ensemble-feature-importance.html



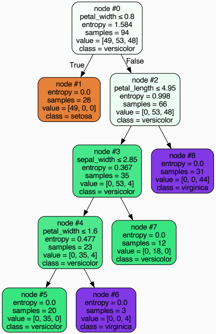
Decision Tree # 4

Notebook: ensemble-feature-importance.html



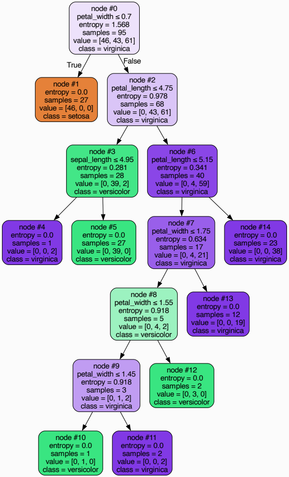
Decision Tree # 5

Notebook: ensemble-feature-importance.html



Decision Tree # 6

Notebook: ensemble-feature-importance.html



node #0
petal_width ≤ 0.7
entropy = 1.568
samples = 95
value = [46, 43, 61]
class = virginica

True

False

node #1
entropy = 0.0
samples = 27
value = [46, 0, 0]
class = setosa

node #2
petal_length ≤ 4.75
entropy = 0.978
samples = 68
value = [0, 43, 61]
class = virginica

node #3
sepal_length ≤ 4.95
entropy = 0.281
samples = 28
value = [0, 39, 2]
class = versicolor

node #6
petal_length ≤ 5.15
entropy = 0.341
samples = 40
value = [0, 4, 59]
class = virginica

node #4
entropy = 0.0
samples = 1
value = [0, 0, 2]
class = virginica

node #5
entropy = 0.0
samples = 27
value = [0, 39, 0]
class = versicolor

node #7
petal_width ≤ 1.75
entropy = 0.634
samples = 17
value = [0, 4, 21]
class = virginica

node #14
entropy = 0.0
samples = 23
value = [0, 0, 38]
class = virginica

node #8
petal_width ≤ 1.55
entropy = 0.918
samples = 5
value = [0, 4, 2]
class = versicolor

node #13
entropy = 0.0
samples = 12
value = [0, 0, 19]
class = virginica

node #9
petal_width ≤ 1.45
entropy = 0.918
samples = 3
value = [0, 1, 2]
class = virginica

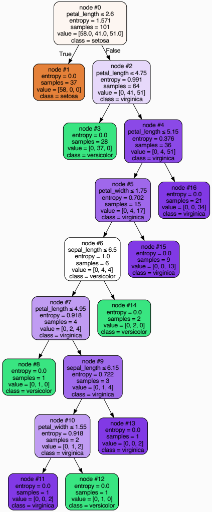
node #12
entropy = 0.0
samples = 2
value = [0, 3, 0]
class = versicolor

node #10
entropy = 0.0
samples = 1
value = [0, 1, 0]
class = versicolor

node #11
entropy = 0.0
samples = 2
value = [0, 0, 2]
class = virginica

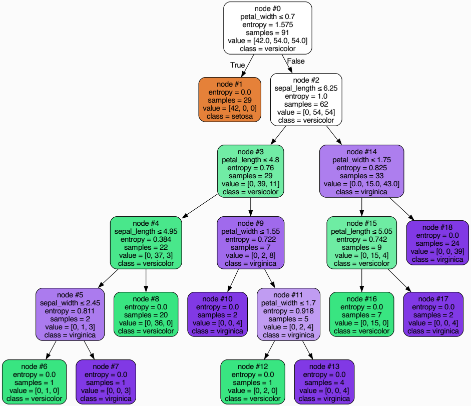
Decision Tree # 7

Notebook: ensemble-feature-importance.html



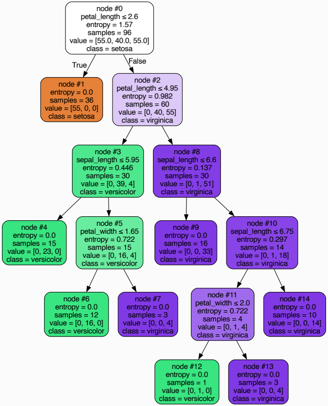
Decision Tree # 8

Notebook: ensemble-feature-importance.html



Decision Tree # 9

Notebook: ensemble-feature-importance.html



Feature Importance: The Big Picture

Definition: What is Feature Importance?

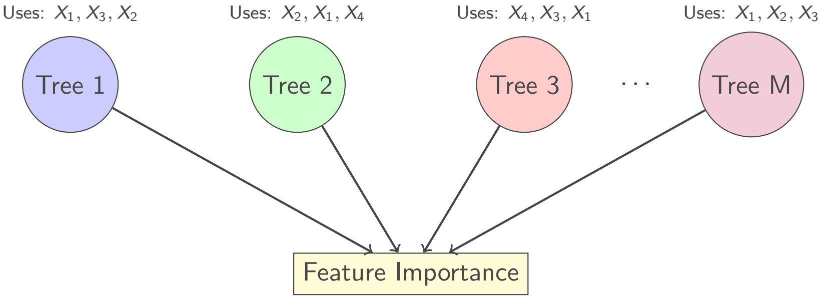
Goal: Measure how much each feature contributes to reducing impurity across all trees in the random forest

Key Points

Core Intuition:

- Features that create “better splits” (reduce impurity more) are more important
- We measure this across ALL trees in the forest
- Then average to get final importance scores

Random Forest: Multiple Trees Overview



Important:

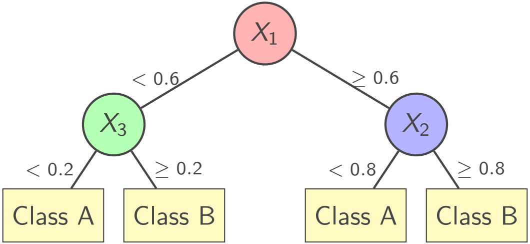
Key Insight: Each tree uses different features at different nodes. We aggregate information from ALL trees!

Step 1: Focus on One Tree (Tree #1)

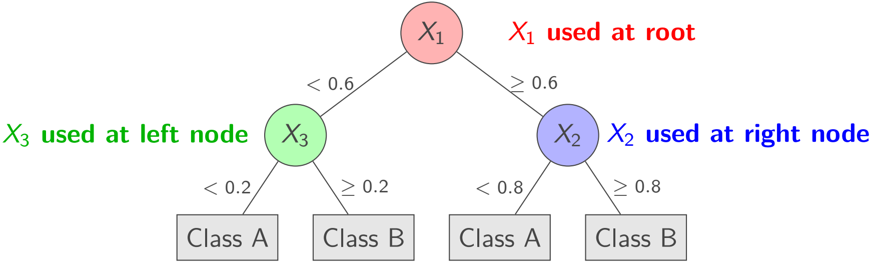
Example:

Let's examine the first tree in our random forest...

Tree #1 Structure



Step 2: Identify Features Used in Tree 1



Key Points

Tree 1 uses: X_1, X_2, and X_3 (but NOT X_4)

Step 3: Focus on Specific Feature ($X_j = X_1$)

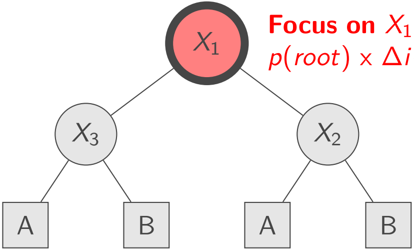
Important:

Let's calculate importance of X_1 **in Tree 1**

Example: F

or X_1 at root node: We need $p(\text{root}) \times \Delta i(\text{root})$

Step 3: Highlighting X_1 in the Tree



Focus on X_1: Calculate $p(root)$ x $\Delta i(root)$

Step 4: Calculate Node Proportion $p(t)$

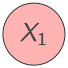
Definition: Node Proportion

$$p(t) = \frac{N_t}{N} = \frac{\text{samples at node } t}{\text{total samples}}$$

Key Points

Root node always has $p(\text{root}) = 1.0$ **(all samples start here)**

Step 4: Visual Example of Node Proportion



1000 samples

Total Dataset: 1000 samples

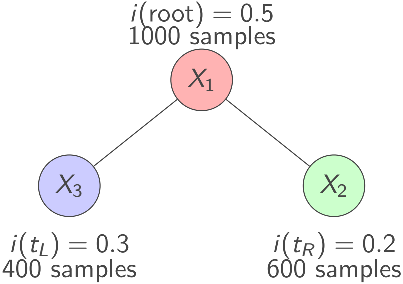
Calculation:

$p(\text{root}) = \frac{1000}{1000} = 1.0$

Step 5: Calculate Impurity Reduction $\Delta i(t)$

Definition: Impurity Reduction Formula

$$\Delta i(t) = i(t) - \frac{N_{t_L}}{N_t} i(t_L) - \frac{N_{t_R}}{N_t} i(t_R)$$



Calculation:

$$\Delta i = 0.5 - \frac{400}{1000} \times 0.3 - \frac{600}{1000} \times 0.2$$
$$= 0.5 - 0.4 \times 0.3 - 0.6 \times 0.2$$
$$= 0.5 - 0.12 - 0.12 = 0.26$$

Step 6: Contribution of X_1 in Tree 1

Example: X_1 Contribution in Tree 1

$$\text{Contribution}_{X_1,\text{Tree 1}} = p(\text{root}) \times \Delta i(\text{root}) = 1.0 \times 0.26 = 0.26$$

Important:

Important: This is just X_1's contribution in ONE tree. We need to do this for ALL trees!

Step 6b: What's Next?

Key Points

Remaining Steps:

- Calculate X_1's contribution in Tree 2, Tree 3, ..., Tree M
- Sum all contributions and divide by M (number of trees)
- This gives us the final importance score for X_1

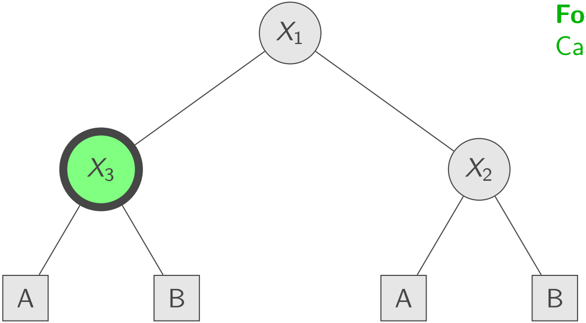
Definition:

General Pattern: Repeat this process for every feature $(X_1, X_2, X_3, X_4, \ldots)$

Now Let's Calculate X_3 Contribution

Important:

Example: Calculate importance of X_3 in the same Tree 1

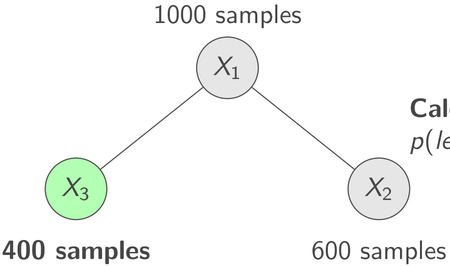


Focus on X_3:
Calculate $p(left) \times \Delta i(left)$

X_3 Step 1: Calculate Node Proportion

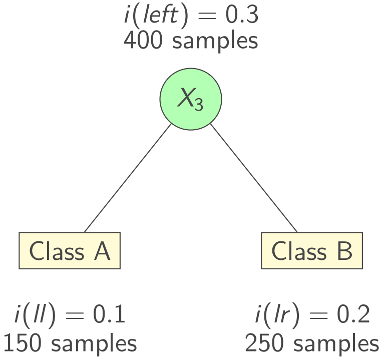
Definition: Node Proportion for Left Child

$$p(left) = \frac{N_{left}}{N} = \frac{\text{samples at left node}}{\text{total samples}}$$



Calculation:
$p(left) = \frac{400}{1000} = 0.4$

X_3 Step 2: Calculate Impurity Reduction



Calculation:

$$\Delta i = 0.3 - \frac{150}{400} \times 0.1 - \frac{250}{400} \times 0.2$$
$$= 0.3 - 0.375 \times 0.1 - 0.625 \times 0.2$$
$$= 0.3 - 0.0375 - 0.125 = 0.1375$$

X_3 Step 3: Final Contribution

Example: X_3 Contribution in Tree 1

$$\text{Contribution}_{X_3,\text{Tree 1}} = p(left) \times \Delta i(left) = 0.4 \times 0.1375 = 0.055$$

Important:

Key Insight: Root nodes typically have higher contributions since $p(root) = 1.0$

Comparison: X_1 vs X_3 in Tree 1

Key Points

Comparison in Tree 1:

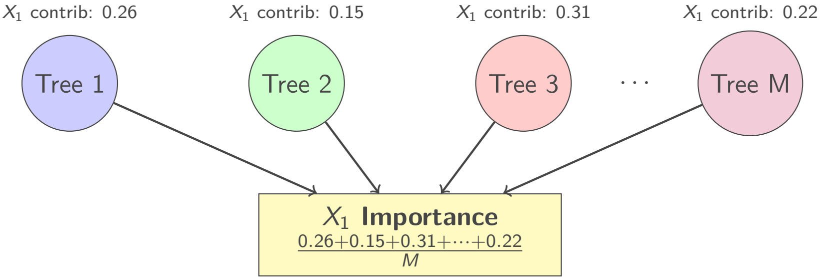
- X_1 contribution: 0.26 (at root)
- X_3 contribution: 0.055 (at left child)

Example:

Why the difference?

- X_1: $p = 1.0$, $\Delta i = 0.26 \rightarrow$ Higher impact
- X_3: $p = 0.4$, $\Delta i = 0.1375 \rightarrow$ Lower impact

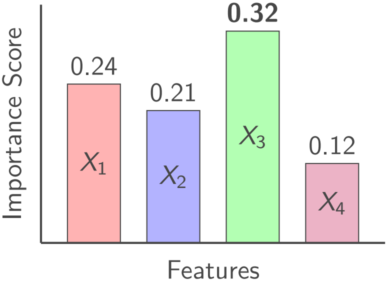
Step 7: Aggregate Across All Trees



Definition: Final Formula for X_1

$$\text{Importance}(X_1) = \frac{1}{M} \sum_{m=1}^{M} \sum_{t \in \text{Tree}_m} \mathbf{1}[j_t = 1] \cdot p(t) \cdot \Delta i(t)$$

Step 8: Repeat for All Features



Key Points

Result: X_3 is most important (0.32), followed by X_1 (0.24), then X_2 (0.21), and X_4 (0.12)

Feature Importance: General Mathematical Formula

Definition: Complete Feature Importance Formula

For any feature X_j in a Random Forest with M trees:

$$\text{Importance}(X_j) = \frac{1}{M} \sum_{m=1}^{M} \sum_{t \in \varphi_m} \mathbf{1}(j_t = j) \cdot p(t) \cdot \Delta i(t)$$

Important:

This formula captures everything we just learned step-by-step!

Formula Components Explained

Key Points

Where:

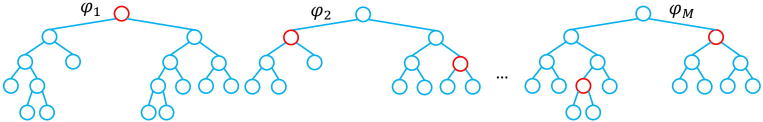
- M = number of trees in the forest
- φ_m = set of all nodes in tree m
- $\mathbf{1}(j_t = j)$ = indicator function (1 if node t uses feature X_j, 0 otherwise)

Example:

Remaining components:

- $p(t) = \frac{N_t}{N}$ = proportion of samples at node t
- $\Delta i(t)$ = impurity reduction at node t

Feature Importance[1]



Importance of variable X_j for an ensemble of M trees φ_m is:

$$\mathrm{Imp}(X_j) = \frac{1}{M} \sum_{m=1}^{M} \sum_{t \in \varphi_m} 1(j_t = j) \Big[p(t) \Delta i(t) \Big],$$

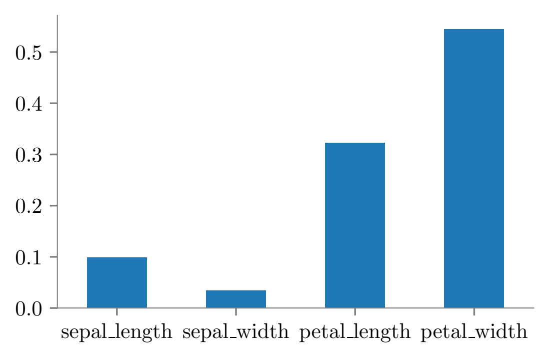
where j_t denotes the variable used at node t, $p(t) = N_t / N$ and $\Delta i(t)$ is the impurity reduction at node t:

$$\Delta i(t) = i(t) - \frac{N_{t_L}}{N_t} i(t_L) - \frac{N_{t_r}}{N_t} i(t_R)$$

[1] Slide Courtesy Gilles Louppe

Computed Feature Importance

Notebook: ensemble-feature-importance.html


sepal_length
sepal_width
petal_length
petal_width